Rehabilitation of the Rio Grande Silvery Minnow

New Mexico
Supercomputing Challenge
Final Report
April 1, 2011

Team 5
AIMS@UNM

Team Members

Nico Ponder

Stefan Klosterman

Louis Jencka

Jake Kileen

Teachers

Terrence Lebeck

Project Mentor

Frederick Bobberts

Table of Contents

Summary

Our model sought to replicate a rehabilitation of the Rio Grande Silvery Minnow in the Rio Grande. As the Rio Grande Silvery Minnow nears extinction, more effort must be made in preserving the species. The Rio Grande Silvery Minnow are an important indicator species, representing the changes dam-building has had upon the fishes' habitat. Biological indicator species are unique environmental barometers as they offer a signal of the biological condition in a watershed. Using bioindicators as an early warning of pollution or degradation in an ecosystem can help sustain critical resources.

Our model has seen multiple iterations, spanning several languages and types. Originally, our program was a math-based model in Java, but its output showed wide and general estimates as well as buggy code. The model then expanded into a command-line interface, written as a .jar, but was still based off of the same buggy code. The final model is a simulation of how the Rio Grande Silvery Minnow could feasibly repopulate in a simulated river, written in C and Objective C.

Our project, in its final incarnation, shows a simulated riverbed, and accounts for areas the fish would favor for eating and reproduction.

Problem Statement

The Rio Grande Silvery Minnow is a prolific producer of fast-hatching eggs, often producing thousands of eggs. These eggs hatch in as little as 3 to 4 days. Because the minnow is so prolific, it once populated a 4,825-kilometer stretch of the Rio Grande, spanning from Colorado to Texas. Just 50 years ago, the minnows' floating eggs were not problematic. Rather, the recent construction of flood-control and river channelization projects began in the 1940s, which coincides with the beginning of the species' decline. The construction projects have converted the Rio Grande into a much narrower and deeper river due to damming projects such as the Cochiti Dam, one of the ten largest of its kind in the United States. As the river sped up and shallow pools gradually disappeared, the minnows' population began to decline. (Ikenson, 2002)

Habitat changes have reduced the minnows' population to an endangered level, and it quickly approaches extinction. Its reproductive strategy, despite the large amount of eggs they produce, is not helping the fishes' population. Its eggs are buoyant, and often drift down the Rio Grande until they are deposited into the waters of Butte Reservoir, where predatory fish may consume the eggs and hatchlings. (Ikenson, 2002)

Rio Grande Silvery Minnow are commonly found in depths < 0.50 meters (m), velocities < 0.40 meters/second (m/sec), and silt substrate, most frequently utilizing debris piles, pool, and backwater mesohabitats. (Torres et al., 2008) However, Elephant Butte's depth is often 20 feet deep (Elephant Butte Reservoir, n.d.,) Cochiti Dam-area water velocity is often greater than 2.5 ft/sec, and contains unsuitable substrate (Torres et al., 2008.) These areas are unusable for the Rio Grande Silvery Minnows' breeding and living conditions.

In the past 70 years, various agencies and states have constructed the Closed Basin Project, the Platoro Dam and Reservoir, the Heron Dam and Reservoir, the El Vado Damn and Reservoir, the Abiquiu Dam and Reservoir, Cochiti Dam and Cochiti Lake, Galisteo Dam and Reservoir, Jemez Canyon Dam and Reservoir, Angosutra Dam and Reservoir, Isleta Diversion Dam, San Acacia Diversion Dam, Elephant Butte Dam and Reservoir, Caballo Dam and Reservoir, Leasburg Dam and Reservoir, American Diversion Dam and American Canal, Amistad Diversion Dam, and the Falcon Diversion Dam. (Fish and Wildlife Service, n.d.)

Because of the large amounts of diversion projects, the Rio Grande Silvery Minnow now cannot move upstream. Around 90 to 95 percent of the minnows' population has been effectively trapped in the 96-kilometer portion of the river downstream of the San Acacia Dam. Furthermore, this stretch of the river is constantly depleted because of diversions for farming and other factors. (Ikenson, 2002)

The Rio Grande Silvery Minnow currently requires a sustained supply of water in the Rio Grande, which is guaranteed not to run dry. They also require a method of moving between segments separated by dams, and a general restoration to a shallow river habitat, especially upstream of San Acacia Dam. They require pre-construction conditions such as shallow water, a low water velocity, and river bottoms of loose sediment (Ikenson, 2002.) With these changes, which will be incorporated into our model, the minnow should be able to properly rehabilitate.

Methods

Initial research on the Rio Grande Silvery Minnow began in September 2010, when one of our group members chose to rehabilitate the Rio Grande Silvery Minnow as a science fair project. The team member collected extensive research, and then presented what can be considered a prototype of our later models. Meanwhile, the team was focused on an alternate model of fluid dynamics, which was unsuccessful. Frederick Bobberts, our former teacher's aide in Chemistry and a former coder at Intel, became our team's mentor, instructing us in proper project design. The model was redesigned, and entered heavy development. After his departure, the team continued development underneath Terrence Lebeck, our Chemistry teacher.

Our method of development focused on the building of one model, which would then be tested, bug-checked, and re-evaluated by the other members of our team. Improvement upon the current model would commence, often in a different coding language, as the output from the current model was measured, checked for feasibility, and charted. When the current model had been deemed bug-free, and its output feasible, the improvement would be checked to see if adjustments made in the current model would need to transition to the current. When the improvement had been modified, it was run multiple times to determine if it had any critical errors. If the improvement had passed all checks, then it would become the current model, and additional improvements would be made.

Improvements were deemed to be variables that made the model more specific and precise. For example, our first model did not account for the fishes' need for a carefully balanced environment, assuming that the fish would be carefully monitored

and cared for, as well as relying on a user to manually input a variable for the starting amount of fish before running the program. The improvement upon that model shifted the format to a command-line .jar, where the user could modify variables without needing to end the program. Improvements from there included reliance on food to survive, ideal water conditions, movement in swarms, and other adjustments necessary to refine the scope of our code.

The current model simulates the fishes' reproduction in a riverbed, divided into "chunks." Chunks help to manage the fishes' movement and population. Similarly, the Rio Grande Silvery Minnow travel in "swarms," our model's term for schools of fish. Within the simulated water conditions, the fish move passively until coming across an "Area of Favorability." In its most basic sense, an Area of Favorability can be described as an area where the water flow is slow enough to sustain the Rio Grande Silvery Minnow, where the depth is neither too shallow or to deep, and where food such as algae can be found.

Areas of Favorability were necessary to create because the Rio Grande Silvery Minnow live in a specifically balanced environment, which can be only slightly modified before the fish encounter deadly problems. The fish are extremely susceptible to environmental changes, and must seek out areas in which they can survive in order to avoid death. Thus, the river is divided into Areas of Favorability, so the fish can seek acceptable living conditions.

Though the fishes can stray from Areas of Favorability, their general health will decline, and the entire swarm will die if it does not return to an Area of Favorability in

time. Similarly, the amount of time since the swarm has encountered a food source has an effect of the swarm's health, causing the swarm to die if the fish cannot eat. The model takes into account the possibility of two swarms meeting, increasing reproductive rates accordingly. This area is where the model had its most trouble, as it takes hours for simulations to complete, and as this segment of code is our longest, it took a significant amount of time to locate where the code was buggy.

Research on the Rio Grande Silvery Minnow

In order to properly replicate the repopulation of the Rio Grande Silvery Minnow, one must first take into account the three to five-year lifespan of the fish, as well as the number of Silvery Minnows in the Rio Grande when they were first declared endangered. In fish, the fertility rate is a representation of the rate at which a fish produces eggs, the quantity of eggs produced, and the percentage that survive adolescence. The lifespan of the fish is the time that the average Silvery Minnow lives. During the process of finding these ratios, the number of females and males, as well as the health of the fish, are taken into account; so they need not be included in this experiment.

In the experiment, the rehabilitation of the Rio Grande Silver Minnow, using the aforementioned variables, will be run as a computer simulation. The program will be experiment-specific code, meaning that its production is the core of the experiment, and will run as a Java program. Java is a programming language published by Oracle, Ltd., its company of origin. The advantages of simulating the experiment include an unlimited number of subjects to test on, rather than the few Silvery Minnows left, and simulations help to prepare for real-world application.

The Rio Grande Silvery Minnow, otherwise known as the *Hybognathus amarus*, populated around 3,800 kilometers of the Rio Grande (*Population and Habitat Viability Assessment,* n.d.) A close relative, the Eastern Silvery Minnow (*Hybognathus regius,*) is also of special concern the state of Massachusetts, but is not yet endangered. The Eastern Minnow has a life span of about three years, becoming sexually mature and able to reproduce by the second year. They breed around April or May during the

daytime, and their eggs take around one week to hatch. Because the Eastern Silvery Minnow is such a close relative to the Rio Grande Silvery Minnow, figures pertaining to the Eastern Minnow could also pertain to the Rio Grande Minnow, and will be treated as such in the experiment (*Eastern Silvery Minnow,* 2008).

The Rio Grande Silvery Minnow is currently endangered and is subject to many threats. For example, according to a recent report by the Natural Heritage & Endangered Species program of Massachusetts, (*Eastern Silvery Minnow*, 2008,) all breeds of Silvery Minnow are threatened by many environmental factors, such as

> Habitat alterations due to increased turbidity, erosion and sedimentation, flow alterations, and pollution are major threats to the Eastern Silvery Minnow. They use aquatic vegetation as habitat and increased turbidity and sedimentation can impact the growth of aquatic vegetation. In addition, sedimentation may cover over quality organic matter that they use for food. Flow alterations can degrade backwater areas critical for spawning.

Due to concern over the threats that Silvery Minnow face, the Natural Heritage & Endangered Species Program of Massachusetts has begun studying the Eastern Minnow.

Despite these threats, the Silvery Minnow was one of the most populous species in the Rio Grande, having lived from Northern New Mexico to the Gulf of Mexico. At the moment, the Rio Grande Silvery Minnow only inhabits one portion of the Rio Grande, the section that runs from Cochiti Dam to Elephant Butte; this is about 7 percent of its former habitat. This extreme decline caused the species to be added to the Endangered Species list in 1994. Until recently, an analysis of the Rio Grande Silvery Minnow's persistence and chance of again becoming populous had not been conducted. (*Population and Habitat Viability Assessment*, n.d.)

The Conservation Breeding Specialist Group was commissioned by the Middle Rio Grande Endangered Species Collaborative Program to produce a population viability analysis (PVA) and strategy to help revive the population of the Rio Grande Silvery Minnow. The PVA, officially called a Population and Habitat Viability Assessment, began development on December 4, 2007. The workshop consisted of thirty experts, who concluded that to keep the Rio Grande Silvery Minnow manageable, the minnows must have adequate short and long-term water supplies, and the scientific community must gather additional information pertinent to the Silvery Minnow to improve the rate of survival through an improved environment. (*Population and Habitat Viability Assessment,* n.d.)

In our model, a growth rate of 2.5% per generation was used, due to this rate being reported in the RIO GRANDE SILVERY MINNOW RECOVERY PLAN, 2010. The viability level, or the point at which the Rio Grande Silvery Minnow would be taken off of the Endangered Species List, was calculated at 240,292 fish using figures and estimates from Dudley & Platania. (2008).

The Code[1]

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <string.h>
#include <pthread.h>

#define Max_Height 15
#define Hour 168 //6 Months
double elapsed;

#pragma mark FISH SWARM

//FISH SWARM
//300 km Max
//2.5 m Average Depth
#define TRUE  1
#define FALSE 0
#define SEED 7654 //Debug ONLY

typedef struct vertex
{
float x,y,z;
} vertex;
typedef struct cartesianCats
{
double x,y,z;
double n, m;
} cartesianCats;
typedef struct algaeArea
{
double x,y,z,n;
double amount;
double quantities;
double timeTillReplenishment;
} algaeArea;
typedef struct Fish {
//Fish statistics - fertility, male:female ratio, resources consumed/required
int dead;
int sex;
float fertilityRate;
float health;
float age;
} Fish;
typedef struct fishSwarm {
//Size of fish swarm. Fish swarms split at x fish.
int fishQuantity;
//Coordinates for center of fish swarm, speed, direction
double x, y, z, magnitude, direction;
//
float avgLifespan;
float timeSinceLastAte;
float averageHealth;
```

[1] The opengl GUI component is omitted here, and included at the end of this report for simplicity.

```
float avgFertilityRate;
float swarmSpawn;
int stationary;
int favorStat;
int split;
Fish *stats;
} fishSwarm;
typedef struct riverChunk
{
cartesianCats *vertexGrid;
algaeArea *favorGrid;
fishSwarm *swarmGrid;
int *swarmCensus;
int *chunkNumber;
int *favorQuantity;
} riverChunk;
typedef struct coordinatePOWNAGE
{
riverChunk *potatoes[10];
int *potatonumber;
} coordinatePOWNAGE;
coordinatePOWNAGE *everything;
cartesianCats *vertexTemp;
int vertexRound;
int swarmMax = 20;

//CODES
#pragma mark ARCHIVIST
#pragma mark OTHER
static void setSeed(float seed)
{
srand(seed);
}

float randomInt()
{
return rand()%20;
}
void THEVOIDDRAWSNEAR()
{
//          free(swarmHive);
//    swarmHive = NULL;
}
#pragma mark RIVER GENERATION
// RIVER GENERATION
void plasmaFractal(int x, int y, float width, float height, float Alpha, float Beta, float Gamma, float Delta, int central)
{

if (width > 2 || height > 1 ) {
float A, B, C, D, center;
center = (Alpha + Beta + Gamma + Delta)/4 + ((.5*(width + height))/(3*(width + height)))*randomInt();
A = .5*(Alpha + Beta);
B = .5*(Beta + Gamma);
C = .5*(Gamma + Delta);
D = .5*(Delta + Alpha);
7;
if(central == 1)
{
if (center > 11.0f)
{
center = 11.0f;
}
```

```
}
if (center > 20) {
center = 20;
}
if (center < 0)
{
center = 0;
}

//<^
plasmaFractal( x, y, (.5*width), (.5*height), Alpha, A, center, D, central);
//<ˇ
plasmaFractal( x, (y+(.5*height)), (.5*width), (.5*height), D, center, C, Delta, central);
//>ˇ
plasmaFractal( (x+(.5*width)), (y+(.5*height)), (.5*width), (.5*height), center, B, Gamma, C, central);
//>^
plasmaFractal( (x+(.5*width)), y, (.5*width), (.5*height), A, Beta, B, center, central);
}
else {
vertexTemp[vertexRound].y = round(z);
vertexTemp[vertexRound].z = y;
vertexTemp[vertexRound].n = (20 - z);
vertexRound++;
}

float z = (Alpha + Beta + Gamma + Delta)/(4 + (.5*(width + height)/3*(width +
height))*(((float)rand()/((float)RAND_MAX))-0.5f));
vertexTemp[vertexRound].x = x;
}

void initRiver(int chunkIdent)
{
//          64x64 gives 4865 vertexes. Eight sections per shunk gives us 38920.
size_t riverSize = sizeof(cartesianCats)*52224;
riverChunk *riverSection = (riverChunk*)malloc(riverSize);
vertexTemp = (cartesianCats*)realloc(vertexTemp, sizeof(cartesianCats)*52224);
vertexRound = 0;
int length = 0;
int tempA, tempB, tempC, tempD, tempE, tempF, cornerA, cornerB;
while (length <= 512) {
if (length == 0) {
tempA = randomInt();
tempB = randomInt();
tempE = randomInt();
tempF = randomInt();
plasmaFractal(length, -32, 32, 32, 20, 20, tempA, tempF, 0);
plasmaFractal(length, 0, 32, 32, tempF, tempA, tempB, tempE, 1);
plasmaFractal(length, 32, 32, 32, tempE, tempB, 20, 20, 0);
cornerA = tempF;
cornerB = tempB;
}
else {
tempC = randomInt();
tempD = randomInt();
plasmaFractal(length, -32, 32, 32, 20, 20, tempC, tempA, 0);
plasmaFractal(length, 0, 32, 32, tempA, tempC, tempD, tempB, 1);
plasmaFractal(length, 32, 32, 32, tempB, tempD, 20, 20, 0);
tempA = tempC;
tempB = tempD;
}

length += 32;
```

```
}
riverSection->vertexGrid = (cartesianCats*)realloc(riverSection->vertexGrid, (sizeof(cartesianCats)*52224));
riverSection->vertexGrid = vertexTemp;

//          BioGen
algaeArea *bioArray = (algaeArea*)malloc(52224*sizeof(algaeArea));
int b = 0;
for (int i = 0; i <= 52224; i++) {
if(vertexTemp[i].n <= 1 || vertexTemp[i].n >= 0)
{
if((rand()%1) < .5)
{
bioArray[b].x = vertexTemp[i].x;
bioArray[b].y = vertexTemp[i].y;
bioArray[b].z = vertexTemp[i].z;
bioArray[b].n = .5*vertexTemp[i].n;
bioArray[b].quantities = 4;
b++;
}
else {
/*                                  bioArray[i].x = vertexTemp[i].x;
bioArray[i].y = vertexTemp[i].y;
bioArray[i].z = vertexTemp[i].z;
bioArray[i].n = vertexTemp[i].n;
*/
}

}
else {
/*                                  bioArray[i].x = vertexTemp[i].x;
bioArray[i].y = vertexTemp[i].y;
bioArray[i].z = vertexTemp[i].z;
bioArray[i].n = pow(vertexTemp[i].n,2);
*/
}
i++;
}
bioArray = (algaeArea*)realloc(bioArray, b*sizeof(algaeArea));
printf("%i favorable vertexes.\n",b);
riverSection->favorQuantity = b;
riverSize += b*sizeof(algaeArea) + sizeof(riverSection->favorQuantity);
riverSection = (riverChunk*)realloc(riverSection, riverSize);
riverSection->favorGrid = (cartesianCats*)malloc(b*sizeof(cartesianCats));
riverSection->favorGrid = bioArray;
fishSwarm *swarmHive = (fishSwarm*)malloc(21*sizeof(cartesianCats)+16000*sizeof(Fish));
swarmHive->stats = (Fish*)malloc(16000*sizeof(Fish));
Fish *fishStats = (Fish*)malloc(2000*sizeof(Fish));

algaeArea random;
int c = b;
for (b = 0; b<=8; b++) {
random = bioArray[rand() %c];
swarmHive[b].fishQuantity = 2000;
swarmHive[b].x = random.x;
swarmHive[b].y = random.y;
swarmHive[b].z = random.z;
swarmHive[b].magnitude = 0;
swarmHive[b].direction = 0;
swarmHive[b].timeSinceLastAte = 0;
int avgRate;
for (int i = 0; i < 2000; i++) {
```

```
fishStats[i].sex = floor(rand()%2);
fishStats[i].fertilityRate = 1.025 + (rand()%1)/100;
avgRate += fishStats[i].fertilityRate;
fishStats[i].health = 100;
}
swarmHive[b].avgLifespan = 4;
swarmHive[b].averageHealth = 100;
swarmHive[b].avgFertilityRate = (avgRate/2000);
swarmHive[b].stats = fishStats;
swarmHive[b].stationary = 0;
}
riverSize += 21*sizeof(cartesianCats) + 2*sizeof(riverSection->chunkNumber);
riverSection = (riverChunk*)realloc(riverSection, riverSize);
riverSection->swarmGrid = swarmHive;

riverSection->chunkNumber = 0;
riverSection->swarmCensus = 8;
char string[20];
char vertexLogChar[20];
char favorLogChar[20];
char swarmLogChar[30];
char saveChar[20];
sprintf(string, "CHUNK%i",chunkIdent);
printf("Archiving Initial River Chunk...\n");
mkdir(string, 0777);
FILE *file;
sprintf(vertexLogChar, "%s/vertex.log",string);
file = fopen(vertexLogChar, "w+");
for (int i = 0; i <= 52224; i++) {
fprintf(file, "(%f,%f,%f)%f ",vertexTemp[i].x,vertexTemp[i].y,vertexTemp[i].z,vertexTemp[i].n);
}
fclose(file);
sprintf(favorLogChar, "%s/favor.log",string);
file = fopen(favorLogChar, "w+");
for (int i = 0; i <= c; i++) {
fprintf(file, "(%f,%f,%f)%f ",bioArray[i].x,bioArray[i].y,bioArray[i].z,bioArray[i].n);
}
fclose(file);

sprintf(swarmLogChar, "%s/swarminit.log",string);
file = fopen(swarmLogChar, "w+");
for (int i = 0; i <= 8; i++) {
fprintf(file, "Swarm %i\n Quantity:%i\nLocation:%f,%f,%f\nVelocity:%f\nOrientation:%f\n\n",i,
swarmHive[i].fishQuantity, swarmHive[i].x, swarmHive[i].y, swarmHive[i].z, swarmHive[i].magnitude,
swarmHive[i].direction);
}
fclose(file);
everything = (coordinatePOWNAGE*)realloc(everything,(riverSize+sizeof(int))*(chunkIdent+2));
//            everything->potatoes = (riverChunk*)realloc(everything->potatoes, riverSize*10);
everything->potatoes[chunkIdent] = riverSection;
everything->potatonumber = chunkIdent+1;
sprintf(saveChar, "%s/binary.log",string);
file = fopen(saveChar, "w+");
fwrite(everything, riverSize, 1, file);

fclose(file);
}

void swarmArchive(fishSwarm *swarm, float time)
{
```

```
}
#pragma mark OVERLORD
//OVERLORD
void swarmIterate(int Time, int Thread)
{

for (int b = 0; b < everything->potatonumber; b++) {
if (everything->potatoes[b]->swarmCensus >= 20 && everything->potatonumber == b-1) {
initRiver(b+1);
}
if (everything->potatoes[b]->swarmCensus >= 20 && everything->potatonumber > b-1) {

}

int c, d;
c = everything->potatoes[b]->swarmCensus;
/*                      if (Thread == 0) {
d = everything->potatoes[b]->swarmCensus;
c = floor(.5*c);
}
else {
d = floor(.5*c);
c = 0;
}
if (Thread == 2) {
c = 0;
d = everything->potatoes[b]->swarmCensus;
}
*/

char string[30];
char string2[30];
char string3[20];
sprintf(string3, "CHUNK%i/swarm",b);
mkdir(string3, 0777);
sprintf(string, "CHUNK%i/swarm/vertex.log",b);
sprintf(string2, "CHUNK%i/swarm/stats.log",b);
FILE *file = fopen(string, "a+");
FILE *stats = fopen(string2, "a+");
cartesianCats *favor = malloc(sizeof(cartesianCats)*52224);
int D = 0;
int xA, yA, zA = 0;
int averageAge;
int averageFert;
int averageHealth;
for (c = 0; c < everything->potatoes[b]->swarmCensus; c++) {
//FISH
/*
for (int A = 0; A <= 52224; A++) {
if(everything->potatoes[b]->vertexGrid[A].x == everything->potatoes[b]->swarmGrid[c].x || everything->potatoes[b]-
>vertexGrid[A].x == everything->potatoes[b]->swarmGrid[c].x + 1 || everything->potatoes[b]->vertexGrid[A].x ==
everything->potatoes[b]->swarmGrid[c].x - 1)
{
if(everything->potatoes[b]->vertexGrid[A].z == everything->potatoes[b]->swarmGrid[c].z || everything->potatoes[b]-
>vertexGrid[A].z == everything->potatoes[b]->swarmGrid[c].z + 1 || everything->potatoes[b]->vertexGrid[A].z ==
everything->potatoes[b]->swarmGrid[c].z - 1)
{
favor[D].x = everything->potatoes[b]->vertexGrid[A].x;
favor[D].y = everything->potatoes[b]->vertexGrid[A].y;
favor[D].z = everything->potatoes[b]->vertexGrid[A].z;
favor[D].n = everything->potatoes[b]->vertexGrid[A].n;
D++;
```

```
}
}
}
/*
*/

for (int A = 0; A < everything->potatoes[b]->favorQuantity; A++) {


if ((everything->potatoes[b]->favorGrid[A].x > everything->potatoes[b]->swarmGrid[c].x || everything->potatoes[b]-
>favorGrid[A].z > everything->potatoes[b]->swarmGrid[c].z)) {
if(sqrt(pow((everything->potatoes[b]->favorGrid[A].x - everything->potatoes[b]->swarmGrid[c].x),2) + pow((everything-
>potatoes[b]->favorGrid[A].z - everything->potatoes[b]->swarmGrid[c].z), 2) <= 20  ))
{
if (everything->potatoes[b]->favorGrid[A].quantities != 0) {
favor[D].x = everything->potatoes[b]->favorGrid[A].x;
favor[D].y = everything->potatoes[b]->favorGrid[A].y;
favor[D].z = everything->potatoes[b]->favorGrid[A].z;
favor[D].n = sqrt(pow((everything->potatoes[b]->favorGrid[A].x - everything->potatoes[b]->swarmGrid[c].x),2) +
pow((everything->potatoes[b]->favorGrid[A].z - everything->potatoes[b]->swarmGrid[c].z), 2));
favor[D].m = A;
D++;
}
}
}
if ((everything->potatoes[b]->favorGrid[A].x < everything->potatoes[b]->swarmGrid[c].x || everything->potatoes[b]-
>favorGrid[A].z > everything->potatoes[b]->swarmGrid[c].z)) {
if(sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - everything->potatoes[b]->favorGrid[A].x),2) + pow((everything-
>potatoes[b]->favorGrid[A].z - everything->potatoes[b]->swarmGrid[c].z), 2) <= 20  ))
{
if (everything->potatoes[b]->favorGrid[A].quantities != 0) {
favor[D].x = everything->potatoes[b]->favorGrid[A].x;
favor[D].y = everything->potatoes[b]->favorGrid[A].y;
favor[D].z = everything->potatoes[b]->favorGrid[A].z;
favor[D].n = sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - everything->potatoes[b]->favorGrid[A].x),2) +
pow((everything->potatoes[b]->favorGrid[A].z - everything->potatoes[b]->swarmGrid[c].z), 2));
favor[D].m = A;
D++;
}
}
}
if ((everything->potatoes[b]->favorGrid[A].x < everything->potatoes[b]->swarmGrid[c].x || everything->potatoes[b]-
>favorGrid[A].z < everything->potatoes[b]->swarmGrid[c].z)) {
if(sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - everything->potatoes[b]->favorGrid[A].x),2) + pow((everything-
>potatoes[b]->swarmGrid[c].z - everything->potatoes[b]->favorGrid[A].z), 2) <= 20  ))
{
if (everything->potatoes[b]->favorGrid[A].quantities != 0) {
        favor[D].x = everything->potatoes[b]->favorGrid[A].x;
        favor[D].y = everything->potatoes[b]->favorGrid[A].y;
        favor[D].z = everything->potatoes[b]->favorGrid[A].z;
        favor[D].n = sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - everything->potatoes[b]->favorGrid[A].x),2)
+ pow((everything->potatoes[b]->swarmGrid[c].z - everything->potatoes[b]->favorGrid[A].z), 2));
        favor[D].m = A;
        D++;
}
}
}
if ((everything->potatoes[b]->favorGrid[A].x > everything->potatoes[b]->swarmGrid[c].x || everything->potatoes[b]-
>favorGrid[A].z < everything->potatoes[b]->swarmGrid[c].z)) {
if(sqrt(pow((everything->potatoes[b]->favorGrid[A].x - everything->potatoes[b]->swarmGrid[c].x),2) + pow((everything-
>potatoes[b]->swarmGrid[c].z - everything->potatoes[b]->favorGrid[A].z), 2) <= 20  ))
{
```

```
		if (everything->potatoes[b]->favorGrid[A].quantities != 0) {
			favor[D].x = everything->potatoes[b]->favorGrid[A].x;
			favor[D].y = everything->potatoes[b]->favorGrid[A].y;
			favor[D].z = everything->potatoes[b]->favorGrid[A].z;
			favor[D].n = sqrt(pow((everything->potatoes[b]->favorGrid[A].x - everything->potatoes[b]-
>swarmGrid[c].x),2) + pow((everything->potatoes[b]->swarmGrid[c].z - everything->potatoes[b]->favorGrid[A].z), 2));
			favor[D].m = A;
			D++;
		}
}
}

/*                                                                      if(sqrt(pow((everything->potatoes[b]-
>favorGrid[A].x - everything->potatoes[b]->swarmGrid[c].x),2) + pow((everything->potatoes[b]->favorGrid[A].z -
everything->potatoes[b]->swarmGrid[c].z), 2) <= 200  ))
//                                                                      {
//                                                                      if (everything->potatoes[b]-
>favorGrid[A].quantities != 0) {
//                                                                      favor[D].x = everything->potatoes[b]-
>favorGrid[A].x;
//                                                                      favor[D].y = everything->potatoes[b]-
>favorGrid[A].y;
//                                                                      favor[D].z = everything->potatoes[b]-
>favorGrid[A].z;
everything->potatoes[b]->swarmGrid[c].x = everything->potatoes[b]->favorGrid[A].x;
everything->potatoes[b]->swarmGrid[c].y = everything->potatoes[b]->favorGrid[A].y;
everything->potatoes[b]->swarmGrid[c].z = everything->potatoes[b]->favorGrid[A].z;
fprintf(file, "%i:%i:(%f,%f,%f)\n",Time,c,everything->potatoes[b]->swarmGrid[c].x,everything->potatoes[b]-
>swarmGrid[c].y,everything->potatoes[b]->swarmGrid[c].z);
return;
//                                                                      }
}
*/
}


/*                                          if (everything->potatoes[b]->swarmGrid[c].x == favor[n].x && everything-
>potatoes[b]->swarmGrid[c].z == favor[n].z) {
//                                                  if (everything->potatoes[b]->favorGrid[]) {

}

/*
everything->potatoes[b]->swarmGrid[i].x = favor[n].x;
everything->potatoes[b]->swarmGrid[i].y = favor[n].y;
everything->potatoes[b]->swarmGrid[i].z = favor[n].z;
A = 8;
*/
int n = 0;
for (int A = 0; A < D; A++) {
if (A == 0) {
n = A;
}
else
{
if (favor[n].n > favor[A].n) {
n = A;
}
else if (favor[n].n == favor[A].n){
if (rand()%1 > .5) {
		n = A;
}
```

```
}
}
}
if(everything->potatoes[b]->swarmGrid[c].stationary == 1 && everything->potatoes[b]->favorGrid[everything-
>potatoes[b]->swarmGrid[c].favorStat].quantities != 0) {
everything->potatoes[b]->favorGrid[everything->potatoes[b]->swarmGrid[c].favorStat].quantities -= 2;

everything->potatoes[b]->swarmGrid[c].timeSinceLastAte = 0;
}
else{
everything->potatoes[b]->swarmGrid[c].stationary = 0;
everything->potatoes[b]->swarmGrid[c].timeSinceLastAte += 1;
if (favor[n].x > everything->potatoes[b]->swarmGrid[c].x) {
xA = favor[n].x - everything->potatoes[b]->swarmGrid[c].x;
}
else {
xA = everything->potatoes[b]->swarmGrid[c].x - favor[n].x;
}
if (favor[n].z > everything->potatoes[b]->swarmGrid[c].z) {
zA = favor[n].z - everything->potatoes[b]->swarmGrid[c].z;
}
else {
zA = everything->potatoes[b]->swarmGrid[c].z - favor[n].z;
}


if (favor[n].y > everything->potatoes[b]->swarmGrid[c].y) {
if (favor[n].x > everything->potatoes[b]->swarmGrid[c].x) {
yA = sqrt(pow((favor[n].y - everything->potatoes[b]->swarmGrid[c].x),2) + pow((everything->potatoes[b]-
>swarmGrid[c].x - favor[n].x), 2));
}
else {
yA = sqrt(pow((favor[n].y - everything->potatoes[b]->swarmGrid[c].x),2) + pow((favor[n].x - everything->potatoes[b]-
>swarmGrid[c].x), 2));

}

}
else
{
if (favor[n].x > everything->potatoes[b]->swarmGrid[c].x) {
yA = sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - favor[n].y),2) + pow((favor[n].x - everything->potatoes[b]-
>swarmGrid[c].x), 2));
}
else
{
yA = sqrt(pow((everything->potatoes[b]->swarmGrid[c].x - favor[n].y),2) + pow((everything->potatoes[b]-
>swarmGrid[c].x - favor[n].x), 2));
}
}


if (favor[n].x > everything->potatoes[b]->swarmGrid[c].x) {
everything->potatoes[b]->swarmGrid[c].x += 1;
//                                          everything->potatoes[b]->swarmGrid[c].y += floor(sin((favor[n].y - everything-
>potatoes[b]->swarmGrid[c].y)/yA));
everything->potatoes[b]->swarmGrid[c].z += 0;
everything->potatoes[b]->swarmGrid[c].favorStat = favor[n].m;
}

if (favor[n].z > everything->potatoes[b]->swarmGrid[c].z) {
everything->potatoes[b]->swarmGrid[c].x += 0;
```

```
//                                        everything->potatoes[b]->swarmGrid[c].y += floor(sin((favor[n].y - everything->potatoes[b]->swarmGrid[c].y)/yA));
everything->potatoes[b]->swarmGrid[c].z += 1;
everything->potatoes[b]->swarmGrid[c].favorStat = favor[n].m;
}
if (everything->potatoes[b]->swarmGrid[c].x  > favor[n].x) {
everything->potatoes[b]->swarmGrid[c].x -= 1;
//                                        everything->potatoes[b]->swarmGrid[c].y += floor(sin((favor[n].y - everything->potatoes[b]->swarmGrid[c].y)/yA));
everything->potatoes[b]->swarmGrid[c].z += 0;
everything->potatoes[b]->swarmGrid[c].favorStat = favor[n].m;
}

if (everything->potatoes[b]->swarmGrid[c].z  > favor[n].z) {
everything->potatoes[b]->swarmGrid[c].x += 0;
//                                        everything->potatoes[b]->swarmGrid[c].y += floor(sin((favor[n].y - everything->potatoes[b]->swarmGrid[c].y)/yA));
everything->potatoes[b]->swarmGrid[c].z -= 1;
everything->potatoes[b]->swarmGrid[c].favorStat = favor[n].m;
}                                       /*
                        everything->potatoes[b]->swarmGrid[c].x += floor(cos(xA/favor[n].n));
                        //                                        everything->potatoes[b]->swarmGrid[c].y += floor(sin((favor[n].y - everything->potatoes[b]->swarmGrid[c].y)/yA));
                        everything->potatoes[b]->swarmGrid[c].z += floor(sin(zA/favor[n].n));
                        everything->potatoes[b]->swarmGrid[c].favorStat = favor[n].m;
                        */

if (everything->potatoes[b]->favorGrid[everything->potatoes[b]->swarmGrid[c].favorStat].x == everything->potatoes[b]->swarmGrid[c].x && everything->potatoes[b]->favorGrid[everything->potatoes[b]->swarmGrid[c].favorStat].z == everything->potatoes[b]->swarmGrid[c].z && everything->potatoes[b]->favorGrid[everything->potatoes[b]->swarmGrid[c].favorStat].quantities != 0) {
everything->potatoes[b]->swarmGrid[c].stationary = 1;
}
if (fmod(Time, Hour) == 0) {
fprintf(file, "%i:%i:(%f,%f,%f)\n",Time,c,everything->potatoes[b]->swarmGrid[c].x,everything->potatoes[b]->swarmGrid[c].y,everything->potatoes[b]->swarmGrid[c].z);
fflush(file);
}

#pragma mark Fish Stats
for (int A = 0; A < everything->potatoes[b]->swarmGrid[c].fishQuantity; A++) {
if (everything->potatoes[b]->swarmGrid[c].stats[A].dead != 1) {

everything->potatoes[b]->swarmGrid[c].stats[A].age += 1;
averageAge += everything->potatoes[b]->swarmGrid[c].stats[A].age;
if (everything->potatoes[b]->swarmGrid[c].stats[A].health == 0) {
        everything->potatoes[b]->swarmGrid[c].stats[A].dead = 1;
        //                                                      return;
}

if (everything->potatoes[b]->swarmGrid[c].stats[A].age >= 1095) {
        if(exp((everything->potatoes[b]->swarmGrid[c].stats[A].age/1000 + rand()%10)) >= 6.5)
        {
                everything->potatoes[b]->swarmGrid[c].stats[A].dead = 1;
                everything->potatoes[b]->swarmGrid[c].fishQuantity -= 1;
        }
}
}
}
if ( 730 <= everything->potatoes[b]->swarmGrid[c].avgLifespan <= 1825 && everything->potatoes[b]->swarmGrid[c].fishQuantity >= 200) {
if(rand()%1000 == 2)
```

```
{
/*int s = everything->potatoes[b]->favorQuantity;
for (c = everything->potatoes[b]->swarmCensus; c<=everything->potatoes[b]->swarmCensus+2; c++) {
		everything->potatoes[b]->swarmGrid[c].fishQuantity = 1500;
		everything->potatoes[b]->swarmGrid[c].x = everything->potatoes[b]->favorGrid[rand()%s].x;
		everything->potatoes[b]->swarmGrid[c].y = everything->potatoes[b]->favorGrid[rand()%s].y;
		everything->potatoes[b]->swarmGrid[c].z = everything->potatoes[b]->favorGrid[rand()%s].z;
		everything->potatoes[b]->swarmGrid[c].magnitude = 0;
		everything->potatoes[b]->swarmGrid[c].direction = 0;
		everything->potatoes[b]->swarmGrid[c].timeSinceLastAte = 0;
		int avgRate;
		for (int i = 0; i < 1500; i++) {
			everything->potatoes[b]->swarmGrid[c].stats[i].sex = floor(rand()%1);
			everything->potatoes[b]->swarmGrid[c].stats[i].fertilityRate = 1.025 + (rand()%1)/100;
//                                                                                  avgRate += everything-
>potatoes[c]->swarmGrid[b].stats[i].fertilityRate;
			everything->potatoes[b]->swarmGrid[c].stats[i].health = 100;
		}
		everything->potatoes[b]->swarmGrid[c].avgLifespan = 4;
		everything->potatoes[b]->swarmGrid[c].averageHealth = 100;
		everything->potatoes[b]->swarmGrid[c].avgFertilityRate = 1.025;
		everything->potatoes[b]->swarmGrid[c].stationary = 0;
}

everything->potatoes[b]->swarmGrid[c].split = 1;
}
}
*/
}
}
averageAge = 0;
averageHealth = 0;
averageFert = 0;

if (everything->potatoes[b]->swarmGrid[c].timeSinceLastAte >= 10) {
for (int A = 0; A < everything->potatoes[b]->swarmGrid[c].fishQuantity; A++) {
		everything->potatoes[b]->swarmGrid[c].stats[A].health -= rand()%3;
}
}
else if (everything->potatoes[b]->swarmGrid[c].timeSinceLastAte == 0) {
for (int A = 0; A < everything->potatoes[b]->swarmGrid[c].fishQuantity; A++) {
if (everything->potatoes[b]->swarmGrid[c].stats[A].health < 100) {
		everything->potatoes[b]->swarmGrid[c].stats[A].health += rand()%10;
}
}
for (int A = 0; A < everything->potatoes[b]->swarmGrid[c].fishQuantity; A++) {
if (averageHealth += everything->potatoes[b]->swarmGrid[c].stats[A].dead != 1) {
averageHealth += everything->potatoes[b]->swarmGrid[c].stats[A].health;
}
}
}
everything->potatoes[b]->swarmGrid[c].averageHealth = (averageHealth / everything->potatoes[b]-
>swarmGrid[c].fishQuantity);
everything->potatoes[b]->swarmGrid[c].avgLifespan = (averageAge / everything->potatoes[b]-
>swarmGrid[c].fishQuantity);
if (fmod(Time, Hour) == 0) {
fprintf(stats, "%i:%i %i %i %i\n",Time,c,everything->potatoes[b]->swarmGrid[c].averageHealth,everything-
>potatoes[b]->swarmGrid[c].fishQuantity,everything->potatoes[b]->swarmGrid[c].swarmSpawn);
fflush(stats);
}
/*                                                  if (fmod(Time, 4383) == 0) {
for (int o; o < floor(1.025*everything->potatoes[b]->swarmGrid[c].fishQuantity); o++) {
```

```
everything->potatoes[b]->swarmGrid[c].stats[o].sex = floor(rand()%2);
everything->potatoes[b]->swarmGrid[c].stats[o].fertilityRate = 1.025 + (rand()%1)/100;
everything->potatoes[b]->swarmGrid[c].stats[o].health = 100;

}

}            */
for (int v = 0; v < everything->potatoes[b]->favorQuantity; v++) {
if (everything->potatoes[b]->favorGrid[v].quantities < 5) {
//                                                                                    everything->potatoes[b]-
>favorGrid[v].quantities += 1;
}
}


}
}
fclose(file);
fclose(stats);
}

}

void iterate()
{
int i = 0;
/*            pthread_t thread1, thread2;
int  iret1, iret2;
int *taskids[2];
*/           while (i <= 50000) {
/*                        taskids[0] = i;
taskids[1] = 0;
iret1 = pthread_create( &thread1, NULL, swarmIterate, (int*)taskids);
taskids[1] = 1;
iret2 = pthread_create( &thread2, NULL, swarmIterate, (int*)taskids);

pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
*/
swarmIterate(i,2);
i++;
}

}
int main (int argc, const char * argv[]) {
initRiver(0);

iterate();
printf("Hello, World!\n");
return 0;
}
```

Results

The data for the project is stored in several separate files and directories, all of which are localized for river 'chunks.'

/CHUNK0 = The main storage for chunk.
/CHUNK0/binary.log = Binary dump data, for resuming simulations.
/CHUNK0/favor.log = The vertex list for the areas of favorability.
/CHUNK0/vertex.log = The river's model file; its vertexes that were derived from the plasma fractal.
/CHUNK0/swarm = The swarm data folder
/CHUNK0/swarm/swarminit.log = A dump of initial swarm data.
/CHUNK0/swarm/stats.log = A swarm's data & statisics.
/CHUNK0/swarm/vertex.log = A chronological list of swarm movements

The two most important files of this data panoply are the "stats.log" and "vertex.log" files found within the "swarm" directory. The 'stats' log stores only the data retaining to a silvery minnow's swarm; its population, the health of its components, the amount of time that has passed since eating or laying eggs, et cetera. "Vertex.log," however, is far more specific, storing only the movements of the swarms within the river.

As showing all the data captured would take up more pages than is truly reasonable and be repetitive, only the most important data for a single 'chunk' run is shown below. This simulation ran about six months worth of one week segments.

In this run, we defined our river to have a 2.5m average depth, an average velocity of 10 km/h, a total possible midpoint height fluctuation of 1.3m, and chunk lengths of 10km. Any deviation, we found in our current model, for the variables of more than 10% results in the extinction of the modeled fish. We've provided only a sampling of the total collected data in order to show its important trends, without flooding the remainder of the report with myriad numbers.[2]

stats.log

1344:1 1452 0 85

1344 = Time in hours
1 = Swarm ID
1452 = Swarm Population
0 = Average Health

[2] The code outputs, on average, over 200 vertexes per second alone, not counting the 'swarm' stats, algae changes, binary dumps, and other miscellaneous data that for our model.

1 = Indicates as to whether the ‘swarm’ was created in the initial chunk creation process or otherwise[3]

0:0 2000 5 1
0:1 2000 51 1
0:2 2000 17 1
0:3 2000 57 1
0:4 2000 68 1
0:5 2000 29 1
0:6 2000 51 1
0:7 2000 81 1
168:2 2000 48 1
168:3 2000 70 1
168:4 2000 39 1
168:5 2000 49 1
168:6 2000 73 1
168:7 2000 53 1
336:1 1996 43 1
336:2 2000 53 1
336:4 1554 83 1
336:5 1851 7 1
336:6 1957 32 1
336:7 1985 22 1
504:2 2000 17 1
504:4 1554 6 1
504:5 1851 91 1
504:6 1957 52 1
504:7 1985 91 1
672:2 2000 75 1
672:4 1554 63 1
672:5 1851 46 1
672:6 1957 23 1
672:7 1985 54 1
840:2 2000 34 1
840:3 2000 84 1
840:4 1554 1 1
840:5 1851 2 1
840:6 1957 40 1
840:7 1985 42 1
1008:0 2000 34 1
1008:1 1996 88 1
1008:4 1554 28 1
1008:5 1851 91 1
1008:6 1957 25 1
1008:7 1985 15 1

[3] Not all of the outputted data has been included here due to size concerns.

1176:0 2000 70 1
1176:1 1996 45 1
1176:4 1554 75 1
1176:5 1851 34 1
1176:6 1957 69 1
1176:7 1985 35 1
1344:2 2000 55 1
1344:3 2000 82 1
1344:4 1554 84 1
1344:5 1851 57 1
1344:6 1957 19 1
1344:7 1985 32 1
1512:0 2000 85 1
1512:1 1996 11 1
1512:2 2000 45 1
1512:3 2000 98 1
1512:4 1554 84 1
1512:5 1851 98 1
1512:6 1957 66 1
1512:7 1985 9 1
1680:0 2000 11 1
1680:1 1996 35 1
1680:2 2000 54 1
1680:3 2000 60 1
1680:4 1554 77 1
1680:5 1851 71 1
1680:6 1957 46 1
1680:7 1985 49 1
1848:0 2000 87 1
1848:1 1996 59 1
1848:2 2000 50 1
1848:3 2000 35 1
1848:4 1554 1 1
1848:5 1851 39 1
1848:6 1957 24 1
1848:7 1985 90 1
2016:0 2000 4 1
2016:1 1996 91 1
2016:2 2000 96 1
2016:3 2000 12 1
2016:4 1554 3 1
2016:5 1851 55 1
2016:6 1957 0 1
2016:7 1985 32 1
2184:2 2000 55 1
2184:3 2000 0 1

2184:4 1554 74 1
2184:5 1851 48 1
2184:6 1957 43 1
2184:7 1985 46 1
2352:2 2000 55 1
2352:3 2000 38 1
2352:4 1554 50 1
2352:5 1851 74 1
2352:6 1957 52 1
2352:7 1985 66 1
2520:2 2000 21 1
2520:3 2000 20 1
2520:4 1554 27 1
2520:5 1851 33 1
2520:6 1957 93 1
2520:7 1985 42 1
2688:2 2000 91 1
2688:3 2000 91 1
2688:4 1554 15 1
2688:5 1851 99 1
2688:6 1957 80 1
2688:7 1985 46 1
2856:2 2000 95 1
2856:3 2000 4 1
2856:4 1554 51 1
2856:5 1851 95 1
2856:6 1957 5 1
2856:7 1985 14 1
3024:2 2000 87 1
3024:3 2000 12 1
3024:4 1554 7 1
3024:5 1851 12 1
3024:6 1957 20 1
3024:7 1985 2 1
3192:0 2000 8 1
3192:1 1996 22 1
3192:2 2000 12 1
3192:3 2000 24 1
3192:4 1554 55 1
3192:5 1851 19 1
3192:6 1957 38 1
3192:7 1985 74 1
3360:2 2000 70 1
3360:3 2000 44 1
3360:4 1554 55 1
3360:5 1851 48 1

3360:6 1957 68 1
3360:7 1985 46 1
3528:0 2000 25 1
3528:1 1996 24 1
3528:4 1554 81 1
3528:5 1851 30 1
3528:6 1957 29 1
3528:7 1985 88 1
3696:0 2000 49 1
3696:1 1996 13 1
3696:4 1554 62 1
3696:5 1851 0 1
3696:6 1957 2 1
3696:7 1985 11 1
3864:0 2000 31 1
3864:1 1996 7 1
3864:4 1554 12 1
3864:5 1851 96 1
3864:6 1957 22 1
3864:7 1985 16 1

vertex.log

336:2:(23.000000,12.000000,-31)

336 = Time in hours
2 = Swarm ID
(23.000000,12.000000,-31) = (x,y,z) of swarm

0:0:(153.000000,10.000000,41.000000)
0:1:(404.000000,16.000000,38.000000)
0:2:(123.000000,12.000000,35.000000)
0:3:(304.000000,14.000000,32.000000)
0:4:(413.000000,8.000000,11.000000)
0:5:(150.000000,17.000000,48.000000)
0:6:(438.000000,11.000000,24.000000)
0:7:(24.000000,19.000000,56.000000)
168:0:(138.000000,11.000000,34.000000)
168:1:(355.000000,14.000000,12.000000)
168:2:(87.000000,12.000000,-9.000000)
168:3:(235.000000,14.000000,-9.000000)
168:4:(301.000000,8.000000,-9.000000)
168:5:(142.000000,17.000000,-8.000000)
168:6:(302.000000,11.000000,-9.000000)
168:7:(134.000000,19.000000,-9.000000)

Analysis & Conclusion

The code ran for a total simulated 161 days. Based upon the rate of growth in the data collected here, we've extrapolated that it would take a total of twenty one years for the fish to reach their original population highpoint of a quarter million, from a starting point of nine thousand fish.

Based upon what data our succession of models has given so far, the Rio Grand Silvery Minnow could be rehabilitated, but only with water levels and speeds at those that existed before waterworks construction. It is a creature that, in our models and research, is shown to be extremely susceptible to the slightest environmental fluctuation. Only through strict control of sections of the Rio Grand, or complete ambandonment of, can bring the Silvery Minnow Back from its current reverie of extinction.

Products of work

Our terrain generation is capable of showing multiple forms of data, as shown in the below photographs. The terrain can be shown as a series of vertices, a series which shows, in color, velocity, or divided between unsafe areas and Areas of Favorability.

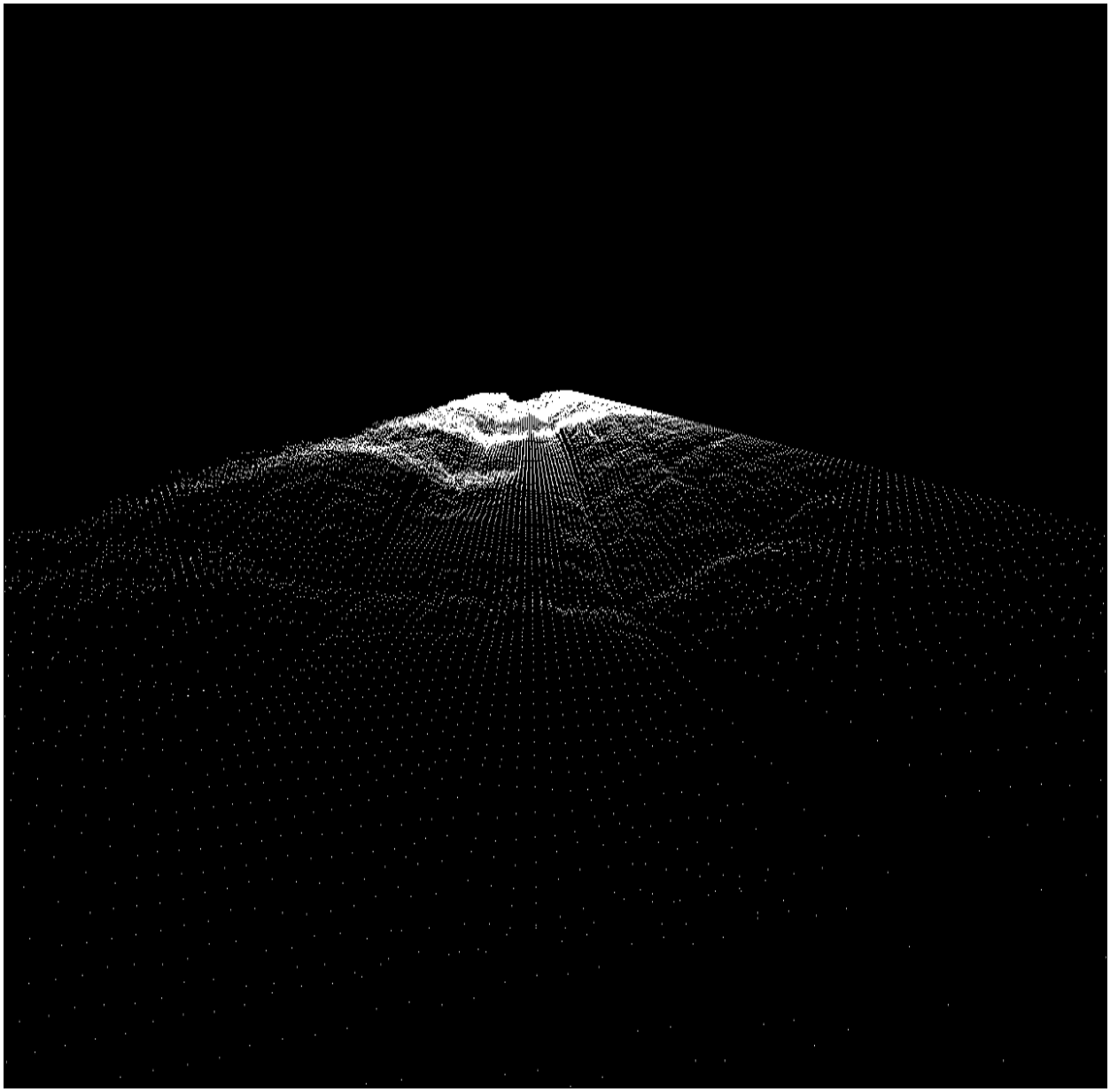

Figure 1- Vertices of our simulated terrain in the river, an accurate representation of the Rio Grande riverbed.

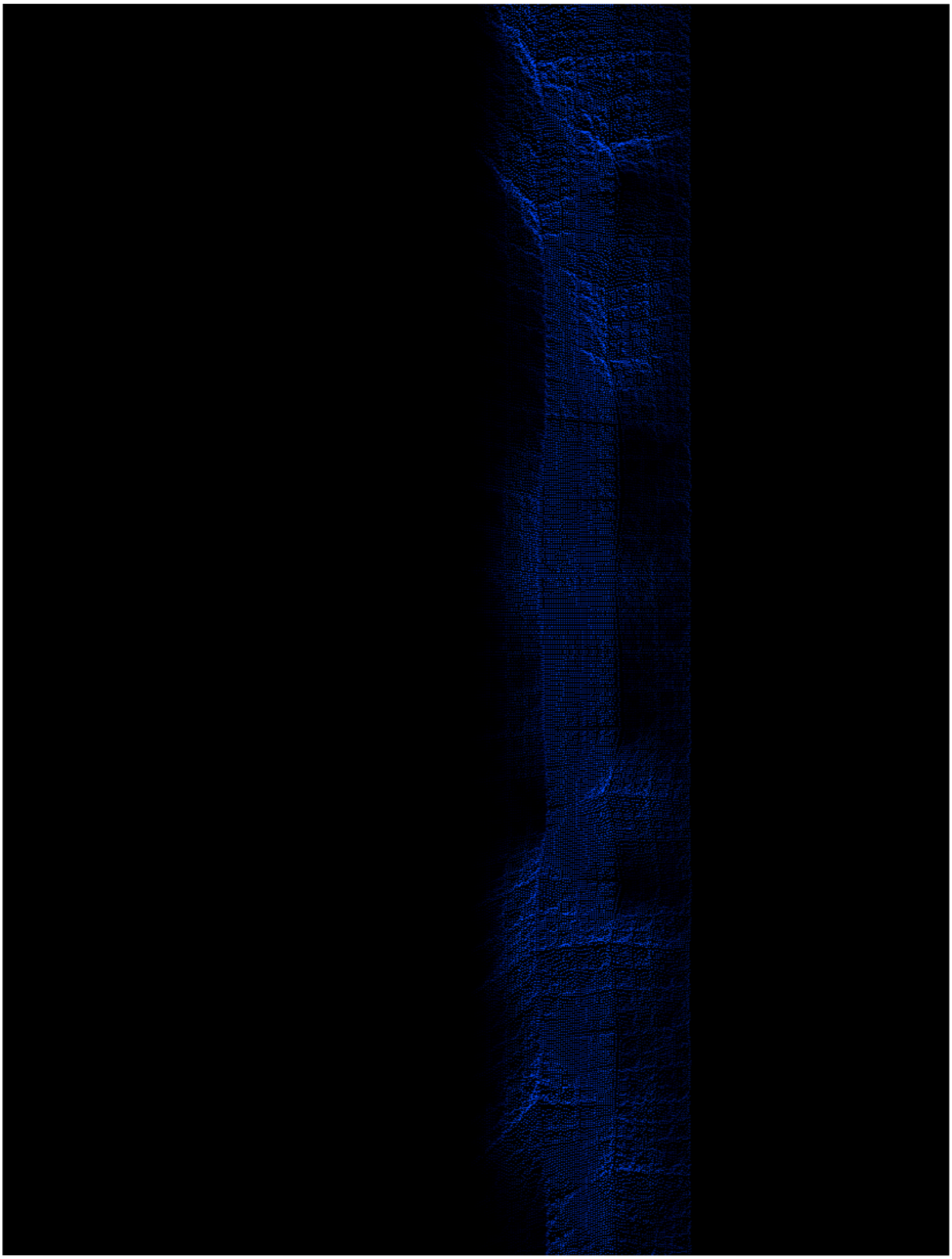

Fig. 2 - A top-down view of a simulated river, where faster velocities are lighter in color, and slower velocities are darker.

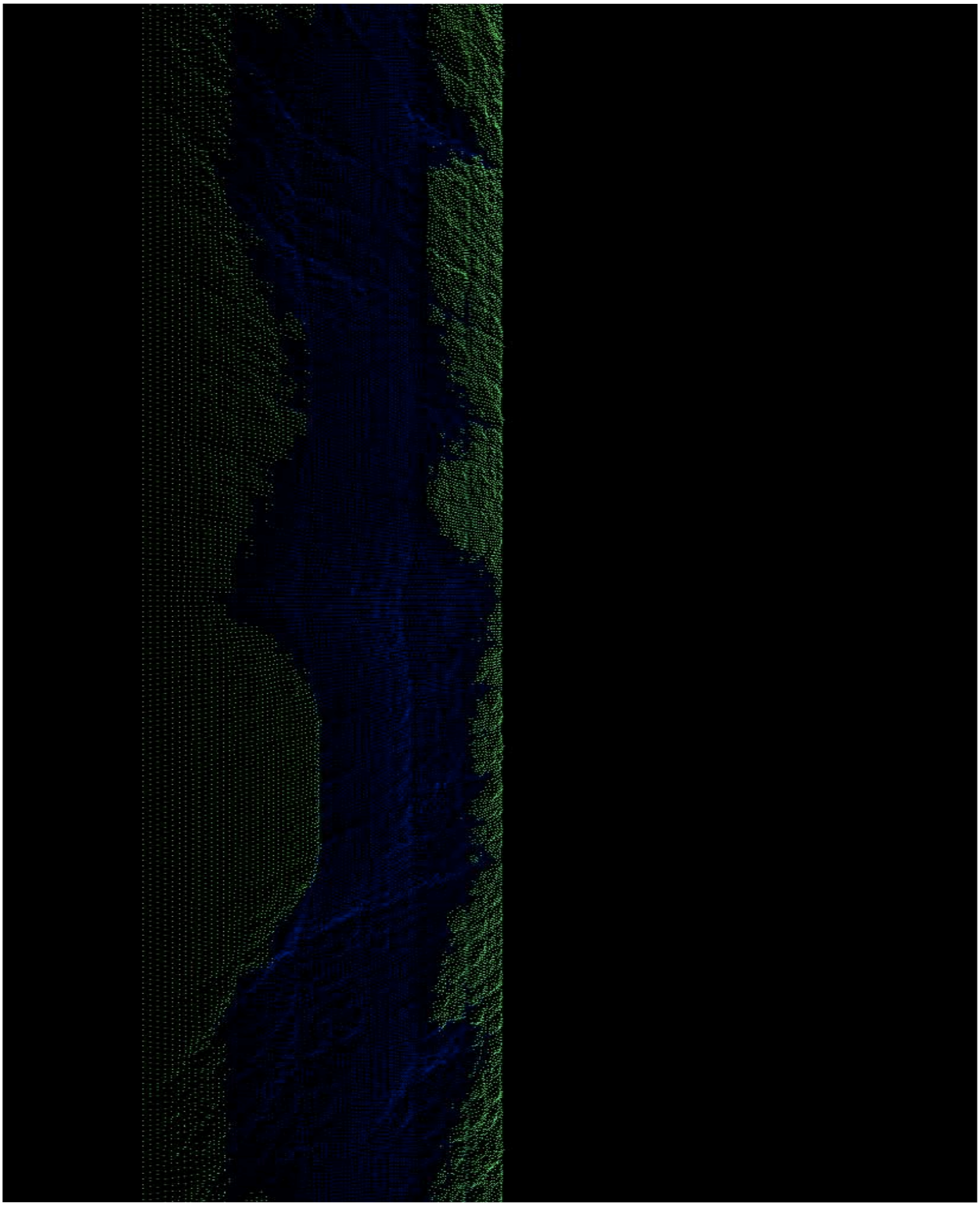

Figure 3- A simulation of the river, showing the Area of Favorability, areas that the Rio Grande Silvery Minnow would prefer. Given that the Rio Grande Silvery Minnow often live on the side of the river, this accurately models their habitat.

Most significant original achievement

Our most significant original achievement is the simulation of the repopulation of the Rio Grande Silvery Minnow in a simulated, flowing riverbed. As far as we have been able to tell, research into repopulation of the Rio Grande Silvery Minnow have been studies of logistics and purely math-based calculations. Our model seems to be the only simulation or study capable of specifically showing how the fish would behave, what they would seek, and how quickly they are capable of reproducing under modeled conditions in the Rio Grande.

References

Alo, D., & Turner, T. (2005, 12 July). Effects of habitat fragmentation on effective population size in the endangered rio grande silvery minnow. *Conservation Biology 19*(4) 1138-1148. Retrieved September 8, 2010, from http://onlinelibrary.wiley.com/doi/10.1111/j.1523-1739.2005.00081.x/abstract

Bestgen, K; Platania. (1991, June). Status and conservation of the Rio Grande Silvery Minnow. S.*The Southwestern Naturalist Vol. 36,* No. 2 (Jun., 1991), pp. 225-232. Retrieved September 8, 2010, from http://www.jstor.org/pss/3671925

Cowley, D. (2006, January). Strategies for ecological restoration of the middle Rio Grande in New Mexico and recovery of the endangered Rio Grande Silvery Minnow. *Reviews in Fisheries Science, 14*(1,2) 169-186. Retrieved September 8, 2010, from www.informaworld.com/smpp/content~db=all~content=a727079144

Dudley, R. K., & Platania, S. P. (2008, August 22). RIO GRANDE SILVERY MINNOW POPULATION MONITORING PROGRAM RESULTS FROM DECEMBER 2006 TO OCTOBER 2007. *middleriogrande.com*. Retrieved November 15, 2010, from http://www.middleriogrande.com/LinkClick.aspx?fileticket=xpW68rQc99Y%3D&tabid=273&mid=680

Eastern Silvery Minnow. (2008, August). *Natural heritage endangered* Mass.gov/dfwele/dfw/nhesp/species_info/nhfacts/hybognathus_regius.pdf

Elephant Butte Reservoir. (n.d.). *Trails.com*. Retrieved December 14, 2010, from http://www.trails.com/tcatalog_trail.aspx?trailid=XFA053-041

Fish and Wildlife Service. (n.d.). Dams and Diversions of the Middle Rio Grande. U.S. Fish and Wildlife Service Home. Retrieved December 14, 2010, from http://www.fws.gov/southwest/mrgbi/resources/dams/index.html

Ikenson, B. (2002). Rio Grande silvery minnow | Endangered Species Bulletin. *BNET*. Retrieved December 14, 2010, from http://findarticles.com/p/articles/mi_m0ASV/is_2_27/ai_90098617/

O'Connor, S. (2002). The rio grande silvery minnow and the endangered

species act. *HeinOnline* . Retrieved September 8, 2010, from http://heinonline.org/HOL/LandingPage?collection=journals&handle=hein.journals/ucollr73&div=23&id=&page=

Rio Grande Silvery Minnow (Hybognathus amarus) Population and Habitat Viability Assessment . (n.d.). *Middle RIo Grande*. Retrieved September 8, 2010, from http://middleriogrande.com/linkclick.aspx?fileticket=dhsypiyi9tc%3D&tabid=324

RIO GRANDE SILVERY MINNOW RECOVERY PLAN. (2010, January 15). *FWS.gov*. Retrieved November 20, 2010, from www.fws.gov/ecos/ajax/docs/recovery_plan/022210_v2.pdf

Torres, L., Remshardt, J., & Kitcheyan, C. (2008, April 17). Habitat Assessment for Rio Grande Silvery Minnow (Hybognathus amarus) in the Cochiti Reach, at Peña Blanca, New Mexico. *Middle Rio Grande Resources*. Retrieved December 14, 2010, from www.middleriogrande.com/LinkClick.aspx?fileticket=PtdypuTTD3k%3D&tabid=468&mid=1159

Acknowledgements

Terrence Lebeck

Frederick Bobberts

AIMS@UNM

Appendix A - The Rest of the Code

The opengl GUI.

```
/*
 *  GLScreen.c
 *  Black
 *
 *
 */

#include "GLScreen.h"

GLuint texture[1]; /* Storage For One Texture ( NEW ) */

GLuint VertexVBOID;
GLuint VertexIVBOID;


static SDL_Surface *gScreen;
GLfloat xpos, ypos, zpos, xrot, yrot;

///
struct VBO
{
    GLfloat x, y, z;        //Vertex
    //        GLfloat nx, ny, nz;     //Normal
    //    float s0, t0;         //Texcoord0
    //    float s1, t1;         //Texcoord1
    //    float s2, t2;         //Texcoord2
    //    GLfloat padding[4];
};

struct Tome
{
    VBO vbo;
    int normaloffset;
    int textureoffset;
    int indiceoffset;
};
typedef struct vertex
{
    float x, y, z;
} vertex;
VBO flashVBO[256];
int flashRound;
float plasmaIteration;

int SEED = 67534;


vertex nA, nB, nC, nD;
///
GLuint objlist;
////
//extern
////


static void initAttributes ()
```

```
{
    // Setup attributes we want for the OpenGL context

    int value;

    // Don't set color bit sizes (SDL_GL_RED_SIZE, etc)
    //    Mac OS X will always use 8-8-8-8 ARGB for 32-bit screens and
    //    5-5-5 RGB for 16-bit screens

    // Request a 16-bit depth buffer (without this, there is no depth buffer)
    value = 32;
    SDL_GL_SetAttribute (SDL_GL_DEPTH_SIZE, value);


    // Request double-buffered OpenGL
    //     The fact that windows are double-buffered on Mac OS X has no effect
    //     on OpenGL double buffering.
    value = 1;
    SDL_GL_SetAttribute (SDL_GL_DOUBLEBUFFER, value);

    xpos = -162.834320f;
    ypos = -118.662239f;
    zpos = -107.850647f;
    xrot = 0.0f;
    yrot = 0.0f;

}

static void printAttributes ()
{
    // Print out attributes of the context we created
    int nAttr;
    int i;

    int  attr[] = { SDL_GL_RED_SIZE, SDL_GL_BLUE_SIZE, SDL_GL_GREEN_SIZE,
        SDL_GL_ALPHA_SIZE, SDL_GL_BUFFER_SIZE, SDL_GL_DEPTH_SIZE };

    char *desc[] = { "Red size: %d bits\n", "Blue size: %d bits\n", "Green size: %d bits\n",
        "Alpha size: %d bits\n", "Color buffer size: %d bits\n",
        "Depth bufer size: %d bits\n" };

    nAttr = sizeof(attr) / sizeof(int);

    for (i = 0; i < nAttr; i++) {
        int value;
            //        SDL_GL_GetAttribute (attr[i], &value);
        printf (desc[i], value);
    }
}

static void createSurface (int fullscreen)
{
    Uint32 flags = 0;

    flags = SDL_OPENGL;
    if (fullscreen)
        flags |= SDL_FULLSCREEN;

    // Create window
    gScreen = SDL_SetVideoMode (1280, 800, 0, flags);
    if (gScreen == NULL) {
```

```
        fprintf (stderr, "Couldn't set 640x480 OpenGL video mode: %s\n",
                 SDL_GetError());
                SDL_Quit();
                exit(2);
    }
}
int LoadGLTextures( )
{
    /* Status indicator */
    int Status = FALSE;

    /* Create storage space for the texture */
    SDL_Surface *TextureImage[1];

    /* Load The Bitmap, Check For Errors, If Bitmap's Not Found Quit */
    if ( ( TextureImage[0] = SDL_LoadBMP( "cow.bmp" ) ) )
    {

        /* Set the status to true */
        Status = TRUE;

        /* Create The Texture */
        glGenTextures( 1, &texture[0] );

        /* Typical Texture Generation Using Data From The Bitmap */
        glBindTexture( GL_TEXTURE_2D, texture[0] );

        /* Generate The Texture */
        glTexImage2D( GL_TEXTURE_2D, 0, 3, TextureImage[0]->w,
                TextureImage[0]->h, 0, GL_BGR,
                GL_UNSIGNED_BYTE, TextureImage[0]->pixels );

        /* Linear Filtering */
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
    }

    /* Free up any memory we may have used */
    if ( TextureImage[0] )
        SDL_FreeSurface( TextureImage[0] );

    return Status;
}
static void loadHeightmap(SDL_Surface *img)
{
}
float randomInt()
{
    return rand()%20;
}

static void plasmaFractal(int x, int y, float width, float height, float Alpha, float Beta, float Gamma, float Delta, int
central)
{

    if (width > 1 || height > 1 ) {
            float A, B, C, D, center;
            center = (Alpha + Beta + Gamma + Delta)/4 + ((.5*(width + height))/(3*(width + height)))*randomInt();
            printf("%f\n",center);
            A = .5*(Alpha + Beta);
            B = .5*(Beta + Gamma);
            C = .5*(Gamma + Delta);
```

```
	D = .5*(Delta + Alpha);
	7;
	if(central == 1)
	{
		if (center < 0)
		{
			center = 0;
		}
		else if (center > 11.0f)
		{
			center = 11.0f;
		}
	}
	if (center > 20) {
		center = 20;
	}

	//<^
	plasmaFractal( x, y, (.5*width), (.5*height), Alpha, A, center, D, central);
	//<ˇ
	plasmaFractal( x, (y+(.5*height)), (.5*width), (.5*height), D, center, C, Delta, central);
	//>ˇ
	plasmaFractal( (x+(.5*width)), (y+(.5*height)), (.5*width), (.5*height), center, B, Gamma, C, central);
	//>^
	plasmaFractal( (x+(.5*width)), y, (.5*width), (.5*height), A, Beta, B, center, central);
}
else {
	float z = (Alpha + Beta + Gamma + Delta)/4;
	plasmaIteration++;
	if (flashRound != 0) {
		if (z >= 15)
		{
			glColor3f(0.0f,1.0f,(20-z)/10);
			glVertex3f(x,z,y);
		}
		else
		{
			glColor3f(0.0f,0.0f,(20-z)/10);
			glVertex3f(x,z,y);
		}
	}

	/*
	 //			printf("%i, %i, %f\n",x,y,z);
	 flashVBO[flashRound].x = x;
	 flashVBO[flashRound].y = y;
	 flashVBO[flashRound].z = z;
	 */
	if(flashRound == 0)
	{
		glBegin(GL_POINTS);

		if (z >= 15)
		{
			glColor3f(0.0f,1.0f,(20-z)/10);
			glVertex3f(x,z,y);
		}
		else {
			glColor3f(0.0f,0.0f,(20-z)/10);
			glVertex3f(x,z,y);

		}
```

```
                nA.x = x;
                nA.y = z;
                nA.z = y;
        }
        if(flashRound == 1)
        {
                nB.x = x;
                nB.y = z;
                nB.z = y;
        }
        if(flashRound == 2)
        {
                nC.x = x;
                nC.y = z;
                nC.z = y;
        }
        if(flashRound == 3)
        {
                nD.x = x;
                nD.y = z;
                nD.z = y;

                vertex V;
                vertex U;
                U.x = nB.x - nA.x;
                U.y = nB.y - nA.y;
                U.z = nB.z - nA.z;

                V.x = x - nA.x;
                V.y = z - nA.y;
                V.z = y - nA.z;

                //              Set Vector U to (Triangle.p2 minus Triangle.p1)
                //              Set Vector V to (Triangle.p3 minus Triangle.p1)

                glNormal3f((U.y * V.z) - (U.z * V.y), (U.z * V.x) - (U.x * V.z), (U.x * V.y) - (U.y * V.x));

                /*              Set Normal.x to (multiply U.y by V.z) minus (multiply U.z by V.y)
                 Set Normal.y to (multiply U.z by V.x) minus (multiply U.x by V.z)
                 Set Normal.z to (multiply U.x by V.y) minus (multiply U.y by V.x)
                 */

                glEnd();
                flashRound = 0;

        }
        else {
                flashRound++;
        }


    }
}
static void drawWorld()
{
    /*
     glBindBuffer(GL_ARRAY_BUFFER, VertexVBOID);
     glVertexPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
     //      glNormalPointer(GL_FLOAT, 64, BUFFER_OFFSET(12));
     glEnableClientState(GL_VERTEX_ARRAY);
     //      glEnableClientState(GL_NORMAL_ARRAY);
```

```
    //      glEnableClientState(GL_INDEX_ARRAY);
    //      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VertexIVBOID);
     glDrawElements(GL_POINTS, 256, GL_FLOAT, 0);
     glDisableClientState(GL_VERTEX_ARRAY);
    //      glDisableClientState(GL_NORMAL_ARRAY);
    //      glDisableClientState(GL_INDEX_ARRAY);
     */
    glCallList(objlist);
}
void normalize(vertex *A, vertex *B)
{
}
void loadOBJ()
{
    objlist = glGenLists(1);
    objLoader *objData = new objLoader();
     objData->load("cow.obj");
    int i = 0;
    int b = 0;
    int count;
    int type;
    count = (objData->faceCount);
    obj_face *f;
     obj_vector *v;

    //      Normals
    vertex V;
    vertex U;
    obj_vector *A;
    obj_vector *B;
    obj_vector *C;

    f = objData->faceList[b];
    if(f->vertex_count == 4)
    {
        type = GL_QUADS;
    }
    else
    {
        type = GL_TRIANGLES;
    }

    printf("Count: %i\n", count);
    glNewList(objlist, GL_COMPILE_AND_EXECUTE);
    //    printf("Vertex Count: %i\nNormals Count: %i\n", count,objData->normalCount);
    while (b < count)
    {
            i = 0;
            f = objData->faceList[b];
            glBegin(type);
            //                      n = objData->normalList[f->normal_index[b]];
            //                      glNormal3f(v->e[0],v->e[1],v->e[2]);
            if (type == GL_TRIANGLES)
            {
                    A = objData->vertexList[f->vertex_index[0]];
                    B = objData->vertexList[f->vertex_index[1]];
                    C = objData->vertexList[f->vertex_index[2]];

                    U.x = B->e[0] - A->e[0];
                    U.y = B->e[1] - A->e[1];
                    U.z = B->e[2] - A->e[2];
```

```
                V.x = C->e[0] - A->e[0];
                V.y = C->e[1] - A->e[1];
                V.z = C->e[2] - A->e[2];

                //              Set Vector U to (Triangle.p2 minus Triangle.p1)
                //              Set Vector V to (Triangle.p3 minus Triangle.p1)

                glNormal3f((U.y * V.z) - (U.z * V.y), (U.z * V.x) - (U.x * V.z), (U.x * V.y) - (U.y * V.x));

                /*              Set Normal.x to (multiply U.y by V.z) minus (multiply U.z by V.y)
                 Set Normal.y to (multiply U.z by V.x) minus (multiply U.x by V.z)
                 Set Normal.z to (multiply U.x by V.y) minus (multiply U.y by V.x)
                 */
        }
        while (i < f->vertex_count)
        {
                v = objData->vertexList[f->vertex_index[i]];
                if (type == GL_QUADS) {
                        glVertex4f(v->e[0],v->e[1],v->e[2],v->e[3]);
                }
                else
                {
                        glVertex3f(v->e[0],v->e[1],v->e[2]);
                }
                //                              printf("Face: %f, %f, %f\n",v->e[0],v->e[1],v->e[2]);
                i++;
        }
        glEnd();
        b++;
   }
    glEnable(GL_CULL_FACE);
   glEndList();



}
static void loadVBO()
{
    /* At moment useless.
     glGenBuffers(1, &VertexVBOID); //Generate VBO
     glBindBuffer(GL_ARRAY_BUFFER, VertexVBOID); //Vertices

     objLoader *objData = new objLoader();
     objData->load("cow.obj");
     int i = 0;
     int b = 0;
     int count;
     int type;
     count = (objData->faceCount);
     obj_face *f;
     obj_vector *v;
     obj_vector *w;

     //      Normals
     vertex V;
     vertex U;
     obj_vector *A;
     obj_vector *B;
     obj_vector *C;

     f = objData->faceList[b];
     if(f->vertex_count == 4)
```

```
    {
    type = GL_QUADS;
    }
    else
    {
    type = GL_TRIANGLES;
    }

    printf("Count: %i\n", count);
    //    printf("Vertex Count: %i\nNormals Count: %i\n", count,objData->normalCount);

    VBO vertices[count*3];
    ushort pindices[count*3];
    while (b < count)
    {
    i = 0;
    //                n = objData->normalList[f->normal_index[b]];
    //                glNormal3f(v->e[0],v->e[1],v->e[2]);
    if (type == GL_TRIANGLES)
    {
    A = objData->vertexList[f->vertex_index[0]];
    B = objData->vertexList[f->vertex_index[1]];
    C = objData->vertexList[f->vertex_index[2]];

    U.x = B->e[0] - A->e[0];
    U.y = B->e[1] - A->e[1];
    U.z = B->e[2] - A->e[2];

    V.x = C->e[0] - A->e[0];
    V.y = C->e[1] - A->e[1];
    V.z = C->e[2] - A->e[2];

    //                Set Vector U to (Triangle.p2 minus Triangle.p1)
    //                Set Vector V to (Triangle.p3 minus Triangle.p1)
    vertices[b].nx = ((U.y * V.z) - (U.z * V.y));
    vertices[b].ny = ((U.z * V.x) - (U.x * V.z));
    vertices[b].nz = ((U.x * V.y) - (U.y * V.x));

    pindices[b] = (f->vertex_index[i]);

    //                Set Normal.x to (multiply U.y by V.z) minus (multiply U.z by V.y)
    //                Set Normal.y to (multiply U.z by V.x) minus (multiply U.x by V.z)
    //                Set Normal.z to (multiply U.x by V.y) minus (multiply U.y by V.x)

    }
    b++;
    }
    for (int i = 0; i < objData->vertexCount; i++)
    {
    w = objData->vertexList[i];
    vertices[i].x = w->e[0];
    vertices[i].y = w->e[1];
    vertices[i].z = w->e[2];
    }

    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW); //Create VBO, 2mb
    glGenBuffers(1, &VertexIVBOID); //Generate VBO
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VertexIVBOID); //Vertices
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(pindices), pindices, GL_STATIC_DRAW); //Create IBO,
2mb
    //  glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(pvertices), &pvertices[0].x);
    //      glDeleteBuffers(1, &VertexVBOID);
```

```
    //      glDeleteBuffers(1, &VertexIVBOID);
    */
}
static void loadModels()
{

    flashRound = 0;
    plasmaIteration = 0;
    /*
     glGenBuffers(1, &VertexVBOID); //Generate VBO
     glBindBuffer(GL_ARRAY_BUFFER, VertexVBOID); //Vertices

     */
    objlist = glGenLists(1);
    glNewList(objlist, GL_COMPILE_AND_EXECUTE);
    //      glBegin(GL_POINTS);
    //      plasmaFractal(<#int x#>, <#int y#>, <#float width#>, <#float height#>, <#float Alpha#>, <#float Beta#>,
<#float Gamma#>, <#float Delta#>)
    float length = 0;
    float tempA, tempB, tempC, tempD, tempE, tempF, cornerA, cornerB;
    while (length <= 512) {
            if (length == 0) {
                    tempA = randomInt();
                    tempB = randomInt();
                    tempE = randomInt();
                    tempF = randomInt();
                    plasmaFractal(length, -32, 32, 32, 15, 15, tempA, tempF, 0);
                    plasmaFractal(length, 0, 32, 32, tempF, tempA, tempB, tempE, 1);
                    plasmaFractal(length, 32, 32, 32, tempE, tempB, 20, 20, 0);
                    cornerA = tempF;
                    cornerB = tempB;
            }
            else {
                    tempC = randomInt();
                    tempD = randomInt();
                    plasmaFractal(length, -32, 32, 32, 15, 15, tempC, tempA, 0);
                    plasmaFractal(length, 0, 32, 32, tempA, tempC, tempD, tempB, 1);
                    plasmaFractal(length, 32, 32, 32, tempB, tempD, 20, 20, 0);
                    tempA = tempC;
                    tempB = tempD;
            }

            length += 32;

            printf("%f\n",((float)rand()*10/((float)RAND_MAX)));

    }

    //      glEnd();
    glEndList();
    printf("%i\n",plasmaIteration);
    /*
     printf("%i",flashRound);
     flashRound = 0;

     glBufferData(GL_ARRAY_BUFFER, sizeof(flashVBO), &flashVBO, GL_STATIC_DRAW); //Create VBO, 2mb
     */
}
static void setSeed(float seed)
{
    srand(seed);
}
```

```
static void initGL ()
{
    NSLog(@"The Aspect Ratio is %@\n",@"1.6");
    glViewport(0, 0, 1280, 800);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //  gluPerspective( 90.0f, 4.0f / 3.0f, 0.1f, 500.0f);
    GLdouble aspect = 4.0f / 3.0f;
    GLdouble fW, fH;
    GLdouble fov = 100.0f;
    fH = tan( (fov / 2) / 180 * pi ) * 0.1f;
    fH = tan( fov / 360 * pi ) * 0.1f;
    fW = fH * aspect;

    glFrustum( -fW, fW, -fH, fH, 0.1f, 500.0f );
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    LoadGLTextures();
    /* Enable Texture Mapping ( NEW ) */
    glEnable(GL_TEXTURE_2D );

    //Normalize
    glEnable(GL_RESCALE_NORMAL);

    //Enable Lighting
    //        glEnable(GL_LIGHTING);

    glEnable(GL_DEPTH_TEST);
}

static void drawGL ()
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glPushMatrix();
    glTranslatef(0,0,-5.0f);
    glRotatef(xrot, 1, 0, 0);
    glRotatef(yrot, 0, 1, 0);
    glTranslatef(xpos,ypos,zpos);
    /*        // Create light components
     glEnable(GL_LIGHT0);
     GLfloat ambientLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };
     GLfloat diffuseLight[] = { 0.8f, 0.8f, 0.8, 1.0f };
     GLfloat specularLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };
     GLfloat position[] = { 0.0f, 0.0f, 0.0f, 1.0f };
     // Assign created components to GL_LIGHT0
     glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
     glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
     glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
     glLightfv(GL_LIGHT0, GL_POSITION, position);
     */      //            glBindTexture( GL_TEXTURE_2D, texture[1] );
    /*                    static double equation[] = {0, 1, 1, 0};
     glEnable(GL_CLIP_PLANE0);
     glClipPlane(GL_CLIP_PLANE0, equation);
     */
    drawWorld();



    glPopMatrix();
```

```
    glPushMatrix();
    glPopMatrix();
    //    glFlush();
    //    SDL_GL_SwapBuffers();


}

static void mainLoop ()
{
    int done = 0;
    int fps = 32;
    int delay = 1000/fps;
    int thenTicks = -1;
    int nowTicks;

    float charxvel;
    float charyvel;
    BOOL upup;            //Whether the up key is up.
    BOOL downup;          //Likewise for the down key.
    BOOL leftup;          //You get the drift.
    BOOL rightup;
    BOOL aup;
    BOOL dup;
    BOOL wup;
    BOOL sup;

    upup = TRUE;          //Initialize all the key booleans to true.
    downup = TRUE;
    leftup = TRUE;
    rightup = TRUE;
    aup = TRUE;
    dup = TRUE;
    wup = TRUE;
    sup = TRUE;



    SDL_Event keyevent;    //The SDL event that we will poll to get events.

    while ( !done ) {

            /* Check for events */


       while (SDL_PollEvent(&keyevent))   //Poll our SDL key event for any keystrokes.
       {
          if (keyevent.type == SDL_KEYDOWN)
          {
             switch(keyevent.key.keysym.sym){
                case SDLK_LEFT:
                   leftup = false;
                   break;
                case SDLK_RIGHT:
                   rightup = false;
                   break;
                case SDLK_UP:
                   upup = false;
                   break;
                case SDLK_DOWN:
                   downup = false;
                   break;
```

```
            case SDLK_a:
                aup = false;
                charxvel = -0.1f;
                break;
            case SDLK_d:
                dup = false;
                charxvel = 0.1f;
                break;
            case SDLK_w:
                wup = false;
                charyvel = -0.1f;
                break;
            case SDLK_s:
                sup = false;
                charyvel = 0.1f;
                break;
            case SDLK_u:
                                                loadModels();
                                                break;
                                        default:
                done=1;
                break;
        }
    }
    if (keyevent.type == SDL_KEYUP)
    {
        switch(keyevent.key.keysym.sym){
            case SDLK_LEFT:
                leftup = true;
                break;
            case SDLK_RIGHT:
                rightup = true;
                break;
            case SDLK_UP:
                upup = true;
                break;
            case SDLK_DOWN:
                downup = true;
                break;
            case SDLK_a:
                aup = true;
                charxvel = 0;
                break;
            case SDLK_d:
                dup = true;
                charxvel = 0;
                break;
            case SDLK_w:
                wup = true;
                charyvel = 0;
                break;
            case SDLK_s:
                sup = true;
                charyvel = 0;
                break;

            default:
                break;
        }
    }
}
```

```
	// Xrot & Yrot

	if (!rightup) yrot += 2.0f;
	if (yrot > 360) yrot = 0;
	if (!leftup) yrot -= 2.0f;
	if (yrot < -360) yrot = 0;

	if (!upup) xrot -= 5.0f;
	if (xrot < -45) xrot = -45;
	if (!downup) xrot += 5.0f;
	if (xrot >45) xrot = 45;



	if (charxvel <= -1 | !dup)
	{
		charxvel = -1;
	}
	else if ( charxvel >= 1 | !aup)
	{
		charxvel = 1;
	}
	else {
		if (!aup) charxvel += 0.1f;
		if (!dup) charxvel -= 0.1f;

	}

	if (charyvel <= -1.5 | !wup)
	{
		charyvel = -1.5;
	}
	else if ( charyvel >= 1.5 | !sup)
	{
		charyvel = 1.5;
	}
	else {
		if (!wup) charyvel -= 0.1f;
		if (!sup) charyvel += 0.1f;
	}

	if (!wup or !sup) {

		xpos += float(sin(yrot / 180 * pi))*charyvel;
		zpos -= float(cos(yrot / 180 * pi))*charyvel;
		ypos -= float(sin(xrot / 180 * pi))*charyvel;


	}

	//	xpos += charxvel;
	//	zpos += -charyvel;


// Draw at 24 hz
//	This approach is not normally recommended - it is better to
//	use time-based animation and run as fast as possible
drawGL ();
SDL_GL_SwapBuffers ();

// Time how long each draw-swap-delay cycle takes
// and adjust delay to get closer to target framerate
```

```
        if (thenTicks > 0) {
            nowTicks = SDL_GetTicks ();
            delay += (1000/fps - (nowTicks-thenTicks));
            thenTicks = nowTicks;
            if (delay < 0)
                delay = 1000/fps;
        }
        else {
            thenTicks = SDL_GetTicks ();
        }

        SDL_Delay (delay);
    }

}

void init()
{
    srand(time(NULL));
    // Set GL context attributes
    initAttributes ();

    // Create GL context
    createSurface (0);

    // Get GL context attributes
    printAttributes ();

    // Init GL state
    initGL ();

    loadModels();

    // Draw, get events...
    mainLoop ();

}

/*
 *  GLScreen.h
 *  Black
 *
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <pthread.h>

#include <SDL/SDL.h>
#include <GLUT/GLUT.h>

#include "obj_parser.h"
#include "objLoader.h"

#define BUFFER_OFFSET(i) ((char *)NULL + (i))
#define Max_Height 15

void init();
```